Python & MicroPython: Programmieren lernen für Einsteiger

GPIOs steuern die Welt

Teil 2

Im ersten Beitrag zum Thema „Python-Programmierung“ wurde die Hardwaresteuerung über die I/O-Pins des Raspberry Pi erläutert. Dabei wurden die Ports lediglich als Ausgänge verwendet, um externe Verbraucher wie z. B. Leuchtdioden anzusteuern. Moderne Prozessoren ermöglichen es jedoch, einen Pin sowohl als Ein- als auch als Ausgang zu konfigurieren. Dadurch können einzelne Pins als universelle Schnittstellen für die gesamte Bandbreite der Elektronik genutzt werden. In diesem Beitrag soll deshalb die Funktion der Pins als Eingänge genauer beschrieben werden. Dabei werden die folgenden Themen und Python-Programmstrukturen vorgestellt: GPIOs als Eingänge; Steuern mit Tastendruck; offene Eingänge und Tasterprellen; einfache Peripheriesteuerung mit „gpiozero“. Die gpiozero Library gestattet eine besonders einfache Ansteuerung und Nutzung der wichtigsten Eingabe/Ausgabe-Funktionen der Raspberry Pi-Pins in Python.

Beschalten von GPIOs

Neben der Möglichkeit, GPIOs als Ein- und Ausgänge zu nutzen, gibt es noch zahlreiche Zusatzfunktionen, die ebenfalls von der Hardware des Raspberry Pi unterstützt werden. Insbesondere Bussysteme wie

- SPI (Serial Peripheral Interface),
- I²C (Inter-Integrated Circuit Bus),
- UART (Universal Asynchronous Receiver/Transmitter),
- PWM (Pulsweitenmodulation) und PCM (Pulse Code Modulation)

bieten eine Fülle von Möglichkeiten zum Anschluss sogenannter peripherer Bausteine.

Die Konfiguration eines I/Os kann entweder beim Starten oder während der Laufzeit eines Programms festgesetzt werden. Die Parallelfunktionen sind gesondert zu aktivieren. Diese erfordern zudem häufig das Laden spezieller Software-Module.

Die I/Os weisen bestimmte Strom- und Spannungsgrenzen auf. Beim Beschalten sollten diese Grenzwerte jederzeit genauestens beachtet werden, da ansonsten nicht nur ein einzelner I/O-Kanal, sondern der gesamte Raspberry Pi beschädigt werden kann. Die exakte maximale Strombelastung der Pins auf einem Raspberry Pi hängt von verschiedenen Faktoren ab, einschließlich des spezifischen Raspberry-Pi-Modells und des verwendeten I/O-Pins. Die GPIO-Pins des Raspberry Pi sind jedoch generell für eine Strombelastung von maximal 16 mA pro Pin ausgelegt. Die gesamte Strombelastung für alle Pins zusammen sollte 50 mA nicht überschreiten.

Die Spannungspegel für GPIO-Ein und -Ausgänge liegen beim Raspberry Pi bei:

- +3,3 V für den „High"-Level (Logisch „1")
- 0 V für den „Low"-Level (Logisch „0")

Genau genommen sind diese Zustände allerdings als Bereiche festgelegt. Deshalb sind hier gewisse Abweichungen zulässig. Je nach Last kann der Pegel an einem GPIO-Ausgang auch von seinen nominalen Werten von 0 V und +3,3 V abweichen.

In der Regel werden GPIO-Eingänge mit Widerständen beschaltet, um sie auf einen definierten Pegel zu setzen (+VCC oder GND) oder um den Strom zu begrenzen. Die Pegel bzw. Spannungswerte unterliegen einer gewissen Toleranz. Ein GPIO-Eingang erkennt Spannungen unter 0,8 V als „Low" und Spannungen über 1,3 V als „High". Dazwischen wird kein definierter Pegel erkannt. In dieser „Lücke" neigen die Eingänge zu unerwünschten Schwingungen. Das heißt, sie nehmen wechselweise den Zustand „High" oder „Low" an. Das passiert zum Beispiel dann, wenn der GPIO-Eingang unbeschaltet ist. Dieser Zustand sollte in der Praxis stets vermieden werden. Über sogenannte Pull-up- bzw. Pull-down-Widerstände kann man Einfluss darauf nehmen, welchen Grundzustand ein GPIO-Eingang haben soll.

GPIO als Eingang (Input)

Ist ein GPIO als Eingang definiert, nimmt er Schaltzustände externer Schaltungsteile an. Das kann im einfachsten Fall ein Taster oder Schalter sein. In jedem Fall muss ein Potential bzw. Pegel (Spannung) anliegen, damit der GPIO-Eingang den anliegenden Zustand auswerten kann. Diese Zustände bezeichnet man als „High" und „Low" oder „1" und „0". Damit drückt man den anliegenden Pegel bzw. die Spannung aus, die nominell +3,3 V bzw 0 V betragen müssen.

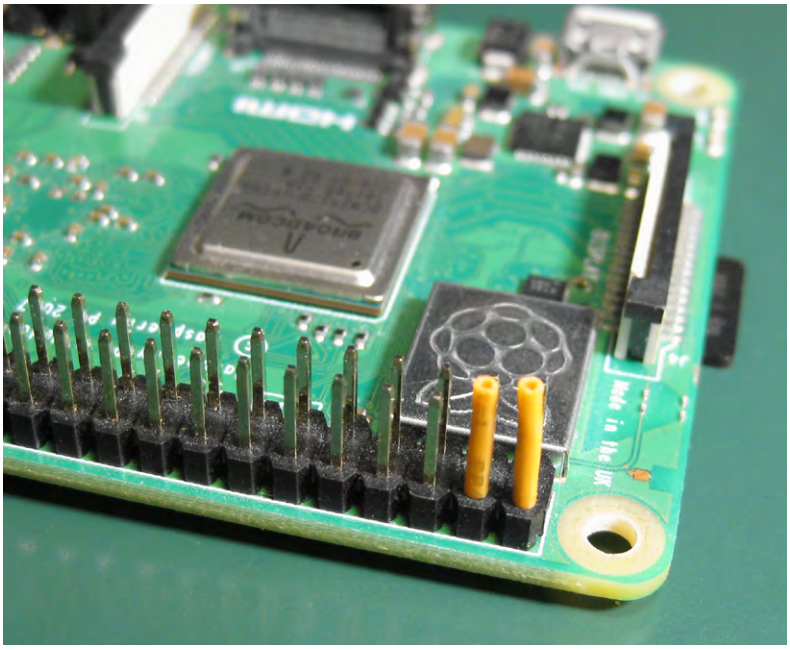


Bild 1: Isolieren der 5-V-Anschlüsse am Raspberry Pi

Wichtig: An einem GPIO-Eingang dürfen niemals mehr als +3,3 V anliegen, da dies den Eingang zerstören würde. Die beiden 5-V-Pins des Raspberry Pi dürfen daher niemals, auch nicht kurzzeitig, mit einem anderen Pin des Raspberry Pi elektrisch verbunden werden. Falls die beiden 5-V-Pins nicht explizit benötigt werden, sollte man sie daher vorsichtshalber isolieren (Bild 1).

Das folgende Programm (open_input.py) demonstriert, wie sich ein offener Eingang auswirkt:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(3, GPIO.OUT)
GPIO.setup(4, GPIO.IN) # pull _ up _ down=GPIO.PUD _ UP)

for i in range(2):
    GPIO.output(3, GPIO.HIGH)
    time.sleep(0.5)
    GPIO.output(3, GPIO.LOW)
    time.sleep(0.5)

while True:
    if GPIO.input(4) == 0:
        GPIO.output(3, GPIO.LOW)
        print(GPIO.input(4))
    else:
        GPIO.output(3, GPIO.HIGH)
        print(GPIO.input(4))
```

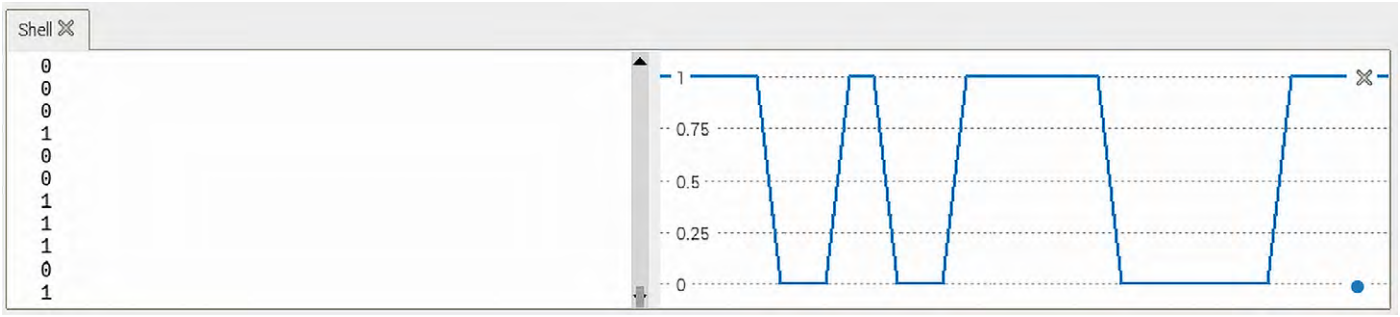


Bild 2: Undefinierter Pegel an einem offenen Eingang

Bleibt der verwendete Eingang unbeschaltet („offen"), wird in der Konsole eine wilde Folge von Nullen und Einsen ausgegeben. Wenn zusätzlich die Plotterfunktion aktiviert wird, zeigt sich auch hier der rasche, undefinierte Pegelwechsel (Bild 2).

Wird an Port 3 das Level-Modul angeschlossen, blinken die rote und grüne LED des gewählten Kanals in unregelmäßiger Folge. Dieses Verhalten kommt zustande, da die offenen Eingänge die immer vorhandenen Störungen aus dem Lichtnetz (50 Hz) oder von Rundfunksendern, WLAN, Bluetooth-Signalen und anderen Hochfrequenzquellen einfangen.

Pull-up- und pull-down-Widerstände

Um einen definierten Eingangspegel zu erhalten, muss man sogenannte Pull-up- oder Pull-down-Widerstände einsetzen. Hierfür können externe Komponenten verwendet werden. Aber auch die Verwendung integrierter Pull-ups/Pull-downs ist möglich. Der ab dem Raspberry Pi 4 verwendete BCM2711-Chipsatz verfügt über interne Pull-up/Pull-down-Widerstände. Diese haben einen Wert von ca. 50 kΩ.

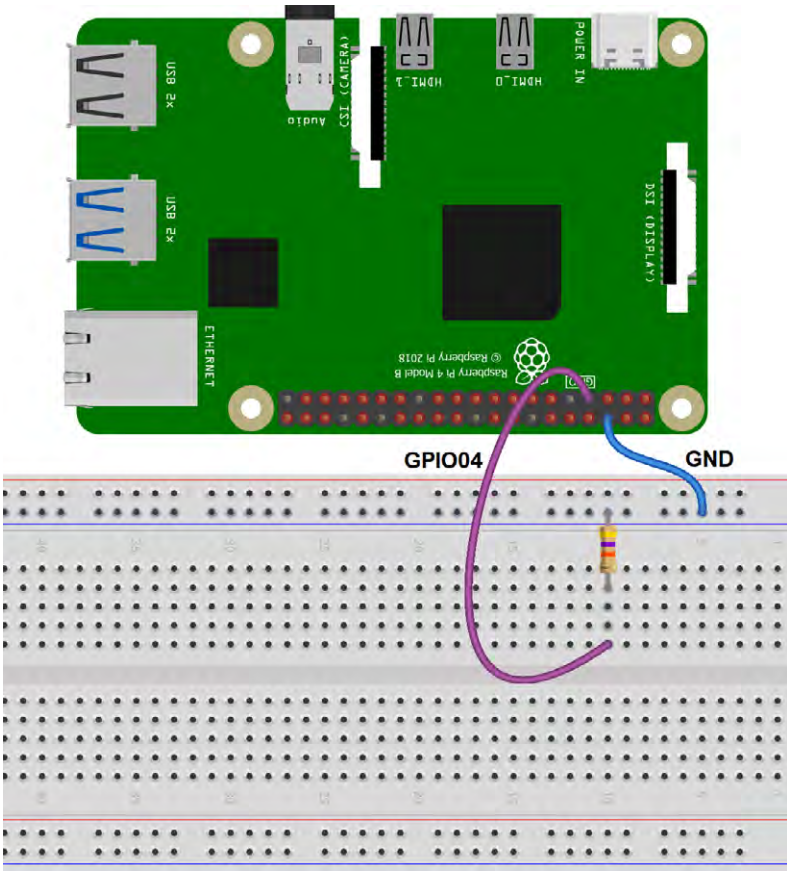


Bild 3: Ein 47-kΩ-Widerstand verbindet Port 4 mit Masse

Mit dem folgenden Programm (pull_up_down.py) wird der interne Pull-up an Port 4 ein- beziehungsweise ausgeschaltet:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

while True:
 GPIO.setup(4, GPIO.IN, pull_up_down=GPIO.PUD_UP)
 time.sleep(1)
 GPIO.setup(4, GPIO.IN)
 time.sleep(1)
```

Nach dem Start des Programms kann man den Eingang (Port 4) mit einem 47-kΩ-Widerstand mit Masse verbinden (Bild 3).

Mithilfe eines Oszilloskops kann der folgende Spannungsverlauf direkt am Pin aufgezeichnet werden (Bild 4).

Aus dem Oszillogramm kann man ablesen, dass der Eingang ohne aktivierten Pull-up-Widerstand auf GND-Potential liegt. Wird der Pull-up aktiviert, geht der Eingang ohne den externen Widerstand erwartungsgemäß auf High Potential (+ 3,3 V). Wenn ein externer 47-kΩ-Widerstand eingeschleift wird, sinkt die Spannung auf etwa 1,6 V (ca. 50 % von + 3,3 V) ab. Daraus ergibt sich, dass der interne Pull-up-Widerstand ebenfalls einen Wert von etwa 40 bis 50 kΩ hat.

Über Pull-up- bzw. Pull-down-Widerstände kann man also Einfluss auf den Grundzustand eines GPIO-Eingangs nehmen. Um den Eingang dann extern umzuschalten, muss man einen Widerstand von deutlich weniger als 50 kΩ verwenden. Ein typisches Vorgehen ist, den internen Pull-up zu aktivieren und dann den Eingang direkt oder über einen 1-kΩ-Widerstand mit einem Schalter oder Taster auf Masse (GND – Ground) zu schalten (Bild 5).

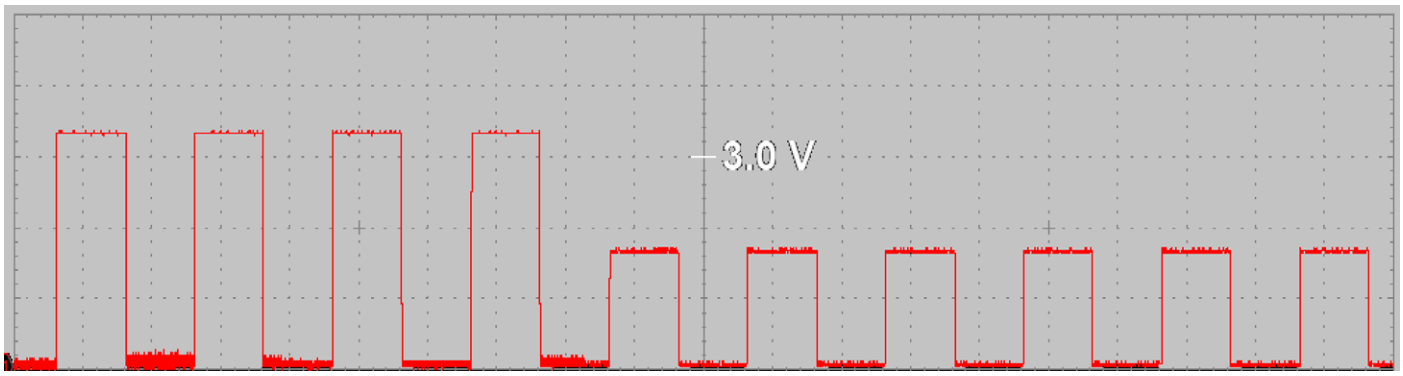


Bild 4: Spannung an einem Eingangskanal

Tastenprellen

Dass aber auch mit Pull-up-/Pull-down-Widerständen nicht immer eindeutige Ergebnisse erzielt werden, zeigt das folgende Programm (bouncing.py):

```
import RPi.GPIO as GPIO
import time

n=0

GPIO.setmode(GPIO.BCM)
GPIO.setup(3, GPIO.OUT)
GPIO.setup(4, GPIO.IN, pull_up_down=GPIO.PUD_UP)

for i in range(2):
 GPIO.output(3, GPIO.HIGH)
 time.sleep(0.5)
 GPIO.output(3, GPIO.LOW)
 time.sleep(0.5)

while True:
 if GPIO.input(4) == 0:
 GPIO.output(3, GPIO.LOW)
 n=n+1
 while GPIO.input(4) == 0:
 pass
 GPIO.output(3, GPIO.HIGH)
 print(n)
```

Das Programm verwendet die RPi.GPIO-Bibliothek, um zwei GPIO-Pins zu steuern. Zunächst werden die notwendigen Bibliotheken – RPi.GPIO für die GPIO-Steuerung und time für Zeitverzögerungen – importiert. Dann wird die Variable „n" initialisiert und auf den Wert 0 gesetzt. Diese Variable dient später zum Zählen der Schaltvorgänge

Anschließend werden die GPIO-Pins des Raspberry Pi konfiguriert. Pin 3 wird als Ausgang (GPIO.OUT) und Pin 4 als Eingang (GPIO.IN) definiert, wobei ein Pull-up-Widerstand aktiviert wird:

pull_up_down=GPIO.PUD_UP

Eine for-Schleife, die zweimal durchlaufen wird, dient als Bereitschaftsanzeige:
Pin 3 wird auf High (+3,3 V) gesetzt, dadurch wird die an diesem Port angeschlossene LED aktiviert. Dann wartet das Programm 0,5 Sekunden. Pin 3 wird auf Low (0 V) gesetzt, und die LED erlischt.

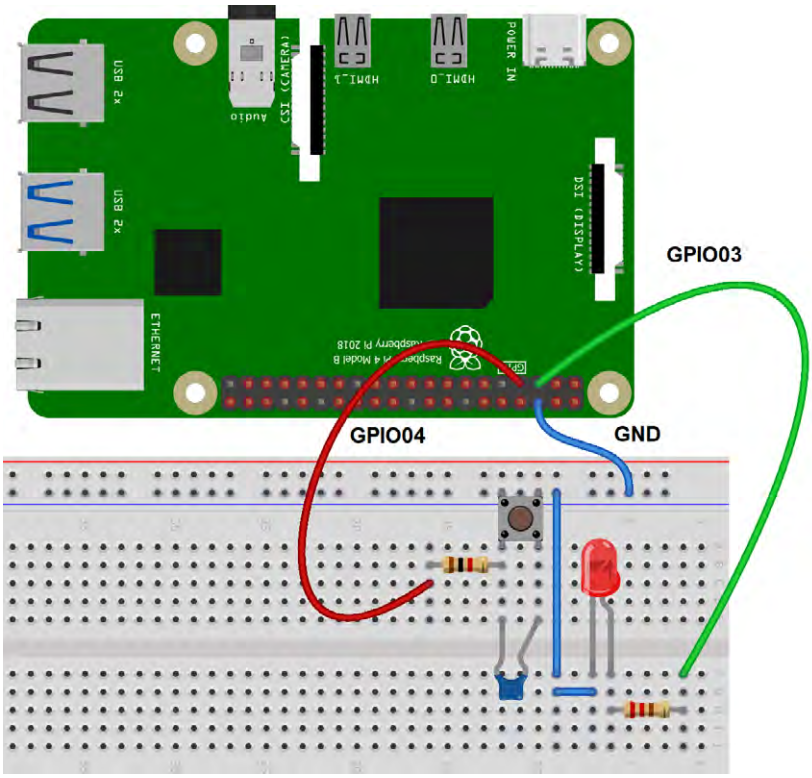


Bild 5: Schalten eines Eingangspins über einen Taster

Das Programm wartet erneut 0,5 Sekunden. Die LED blinkt also genau 2x auf und zeigt so die Bereitschaft der Schaltung an.

Es folgt eine Endlosschleife (while True), die auf Änderungen am Zustand von Pin 4 wartet:
Wenn Pin 4 auf Low (0 V) gesetzt wird, was durch das Drücken des Tasters geschieht, wird die LED an Pin 3 auf Low gesetzt, um sie zu deaktivieren. Danach wird die Variable „n" um 1 erhöht.

Das Programm wartet dann, bis der Taster losgelassen wird, indem es eine Schleife durchläuft, solange Pin 4 auf Low bleibt.

Sobald der Taster losgelassen wird und Pin 4 wieder auf High-Potential liegt, wird die LED wieder eingeschaltet und der aktuelle Wert von „n" auf der Konsole ausgegeben.

Das Programm ermöglicht es also, Tastenbetätigungen zu zählen und die Anzahl auf der Konsole auszugeben, während eine LED jedes Mal aktiviert wird, wenn die Taste gedrückt wird. Es kann somit verwendet werden, um einfache Schalter- oder Tasteninteraktionen zu zählen. Würde man den Taster mit einer Schranke kombinieren, könnte man damit z. B. automatisch die Anzahl von Fahrzeugen oder Personen zählen, die die Schranke passieren.

Die Variable „n" sollte bei jedem Betätigen des Tasters um genau den Wert 1 erhöht werden. Beim Testen des Programms fällt jedoch auf, dass „n" oft um den Wert 2, 3 oder sogar mehr „springt". Ursache ist das sogenannte „Prellen" des Tasters. Aufgrund mechanischer Effekte schließt dieser nicht immer sofort und eindeutig bei seiner Betätigung. Vielmehr wird zunächst der Kontakt geschlossen, dann durch interne Federeffekte wieder kurz geöffnet und anschließend erneut geschlossen, bis sich der endgültige Zustand einstellt. Jedes Öffnen und Schließen wird jedoch vom Raspberry Pi einzeln erfasst. Bild 6 zeigt diesen Vorgang als Oszillogramm.

Um das unerwünschte Prellen zu unterdrücken, kann man einen Kondensator (z. B. 100 nF, siehe Bild 5) parallel zum Schalter schalten. Der so entstehende Tiefpass unterdrückt die einzelnen Schaltimpulse.

Alternativ können auch softwareseitig Vorkehrungen getroffen werden, um ein klares Schaltsignal zu erhalten. Eine Methode ist, den Schaltzustand erst nach einer gewissen Wartezeit auszuwerten, d. h., so lange zu warten, bis das endgültige Spannungspotential erreicht wurde. Weitere Details hierzu werden in späteren Artikeln erläutert.

Übungen und Erweiterungen

- Wie verhalten sich verschiedene Tastertypen hinsichtlich des Prellens?
- Welcher Kondensatorwert liefert die besten Schaltsignale?

Die gpiozero Library

„gpiozero" ist eine Python-Bibliothek, die eine einfache Schnittstelle für GPIO-Ports bietet. Die Bibliothek ermöglicht es, schnell und unkompliziert mit der Programmierung von elektronischen Komponenten zu beginnen. Sie bietet eine abstrakte und benutzerfreundliche Oberfläche, die es Entwicklern ermöglicht, Bauelemente wie Taster, LEDs, Motoren und Sensoren anzusteuern.

Die Library bietet eine Fülle an Möglichkeiten:

- GPIO-Pins ein- und ausschalten
- Ereignisse überwachen
- auf Tastendrücke reagieren
- PWM (Pulse Width Modulation = Pulsweitenmodulation) für die Steuerung von LEDs und Motoren verwenden
- und vieles mehr

Die Bibliothek vereinfacht die Komplexität der GPIO-Programmierung und abstrahiert viele Details, sodass Entwickler sich auf die eigentliche Funktionalität ihrer Projekte konzentrieren können.

Die gpiozero ist in der Python-Standardbibliothek enthalten. Das bedeutet, dass sie nicht separat installiert werden muss. Sie kann direkt in Python-Projekten verwendet werden, indem sie über

```
import gpiozero
```

am Anfang des Codes eingefügt wird.

Einfaches Ansteuern von Komponenten

Um beispielsweise die Button-Schnittstelle von gpiozero zu verwenden, muss diese lediglich importiert werden:

```
from gpiozero import Button
```

Mit diesem Import ist dann die Button-Klasse in jedem Skript verwendbar. Die Button-Klasse stellt eine einfache Schnittstelle zum Lesen von Tasten bereit.

Eine Instanz der Button-Klasse wird via

```
button = Button(4)
```

erstellt. Die Methode „is_pressed" erlaubt es nun, den Status eines Tasters abzurufen. Auch Ereignisse wie „when_pressed" oder „when_released" stehen nun zur Verfügung, um auf Tastendrücke zu reagieren.

Alternativ kann die gesamte gpiozero-Bibliothek importiert werden:

```
import gpiozero
```

In diesem Fall müssen alle Verweise auf Elemente innerhalb von gpiozero mit einem Präfix versehen werden:

```
button = gpiozero.Button(2)
```

Das aus dem ersten Beitrag bekannte Programm zum periodischen Schalten einer LED kann ebenfalls mit gpiozero vereinfacht werden.

```
from gpiozero import LED
from time import sleep

red = LED(17)

while True:
    red.on()
    sleep(1)
    red.off()
    sleep(1)
```

Es geht sogar noch einfacher:

```
from gpiozero import LED
from signal import pause

red = LED(17)
red.blink()
pause()
```

Dabei kann die blink()-Anweisung über zwei Parameter gesteuert werden:

```
blink(onTime, offTime)
```

wobei die Ein- bzw. Ausschaltzeit in Sekunden angegeben werden muss. Mit

```
blink(1, 2)
```

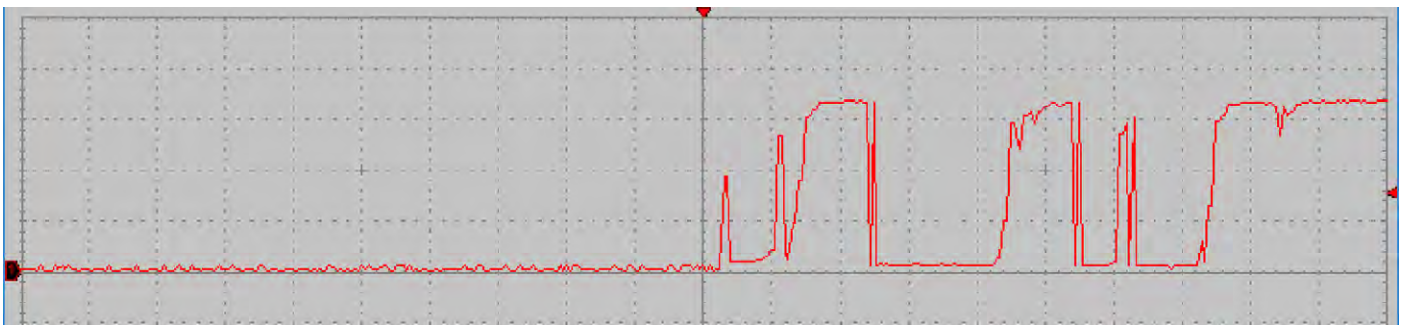


Bild 6: Tasterprellen

wird die LED also 1 Sekunde lang ein und 2 Sekunden lang ausgeschaltet. Werden keine Parameter angegeben, kommen die default-Werte:

onTime = 1
offTime = 1

zur Anwendung.

Bei Verwendung einer Python-Shell (Thonny, IPython oder IDLE) kann die Anweisung pause() entfallen. Wird das Skript jedoch als Python-Datei gespeichert und ausgeführt, würde die LED nur kurz aufblinken, das Skript würde enden und die LED würde sich ausschalten.

Bei Verwendung der pause()-Anweisung wird das Skript dagegen permanent ausgeführt, bis es manuell beendet wird (z. B. durch Drücken von Strg+C).

Auch die Helligkeit einer LED kann so gesteuert werden:

```
from gpiozero import PWMLED
from time import sleep

led = PWMLED(17)

while True:
    led.value = 0  # off
    sleep(1)
    led.value = 0.5  # half brightness
    sleep(1)
    led.value = 1  # full brightness
    sleep(1)
```

Sogar fließende Übergänge sind möglich:

```
from gpiozero import PWMLED
from signal import pause

led = PWMLED(17)
led.pulse()
pause()
```

Analog zu blink() kann auch pulse über zwei Parameterwerte gesteuert werden:

Pulse(rampUp, RampDown)

Wobei die rampUp die Zeit zum Hellerwerden, rampDown entsprechend für das Abdunkeln angibt. Bild 7 zeigt, wie das original PWM-Signal direkt an der LED sowie das Tiefpass-gefilterte Quasi-Analogsignal aussieht.

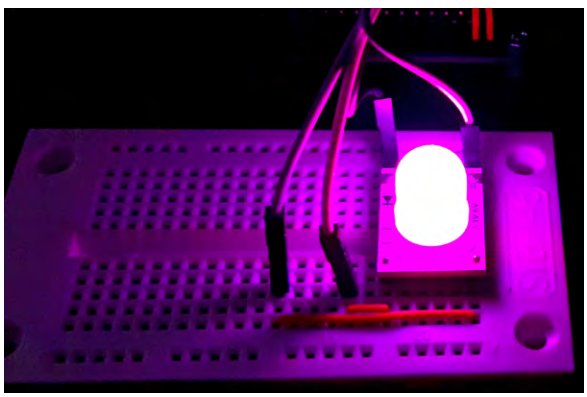
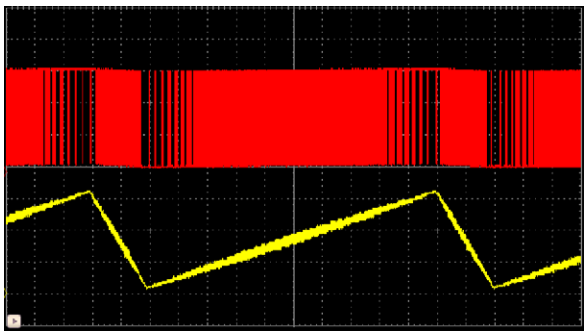


Bild 8: PWM-gesteuerte Dreifarben-LED

Bild 7: PWM-Signal mit led.pulse(5,1)



Multicolor-LED als Stimmungslicht

Besonders beeindruckend sind fließende Übergänge, wenn eine Multicolor-Leuchtdiode verwendet wird. Diese Bauelemente werden auch als RGB-LEDs bezeichnet, da sie jeweils eine rote (R) eine grüne (G) und eine blaue (B) LED integriert haben.

Das folgende Programm (PWM_LED.py) erzeugt ein buntes Farbenspiel (Bild 8):

```
from gpiozero import PWMLED
from signal import pause

led_red = PWMLED(17)
led_red.pulse(2)

led_green = PWMLED(27)
led_green.pulse(4)

led_blue = PWMLED(22)
led_blue.pulse(8)
pause()
```

Fazit und Ausblick

Im zweiten Teil des Python-Grundkurses wurde das Ansteuern von Ports mithilfe von Python erläutert. Mit der gpiozero-Bibliothek können diese Aufgaben auf sehr einfache Weise bewältigt werden. Python wird dadurch noch benutzerfreundlicher, und der Anschluss externer Komponenten wie LEDs, Tastern und Ähnlichem wird zum Kinderspiel.

Im nächsten Beitrag werden die logischen Funktionen von Python genauer untersucht. Diese bilden die Grundlage für alle komplexeren Aufgaben im Bereich von Abfragen und Entscheidungen usw. Dabei wird nicht nur die Software-Seite betrachtet, sondern es werden auch Hardware-Komponenten eingesetzt, die den Anwendungsbereich des Raspberry Pi und seiner Standard-Programmiersprache Python wesentlich erweitern. ELV

Weitere Infos

[1] G. Spanner, MicroPython, Elektor Verlag (2020)
[2] G. Spanner, Raspberry Pi 4 und Pico – Pfiffige Projekte zum Messen, Steuern und Regeln, Elektor Verlag (2023)

Download-Paket

Material	Artikel-Nr.
z. B. Raspberry Pi 4 Model B, 8GB RAM	250567
Experimentier-/Steckboard EXSB1	153753
LEDs mit Vorwiderständen sind z. B. im PAD2 (linear) enthalten	154712